LCD01G Features

LDPC Decoder

- GEO Mobile Radio (GMR–1) compatible
- Nominal code rates of 1/2, 2/3, 3/4, 4/5 and 9/10
- Data lengths from 488 to 8880 bits
- Optional QPSK, 16APSK and 32APSK demapping, descrambling and deinterleaving
- Includes ping–pong input and output memories
- Up to 248 MHz internal clock
- Up to 143 Mbit/s with 30 decoder iterations
- 8–bit log–likelihood–ratio (LLR) or 9–bit inphase and quadrature two's complement input data
- Programmable signal points
- Up to 256 iterations
- Scaled min–sum modified Gauss–Seidel decoding algorithm
- Optional power efficient early stopping
- Parity check output
- Xilinx 7–Series: 19,008 LUTs, 55 18KB BlockRAMs. Altera Stratix–II: 17,853 ALUTs, 168 M4K; Cyclone IV: 30,930 LEs, 120 M9K; Cyclone V: 17,435 ALUTs, 121 M10K.
- Free simulation software

- Available as EDIF core and VHDL simulation core for Xilinx 7–Series FPGAs under SignOnce IP License. Actel, Altera, Lattice and older Xilinx FPGA cores available on request.
- Available as VHDL core for ASICs

Introduction

The LCD01G is a fully compatible GEO Mobile Radio (GMR–1) LDPC [1] error control decoder. Nominal code rates are 1/2, 2/3, 3/4, 4/5 and 9/10. Irregular repeat accumulate quasi–cyclic LDPC codes with weight 1 square circulants in the parity check matrix are used. Circulant sizes range from 16x16 to 87x87. There are from 6 to 40 circulant rows and from 16 to 150 circulant columns in the parity check matrix.

The check node degree is irregular with two different values for each code. Due to the accumulate encoding method for the parity check bits, the first row has a check degree which is one less than all the other rows. The first row has a check degree ranging from 6 to 36 with the other rows having a check degree which is one higher (7 to 37).

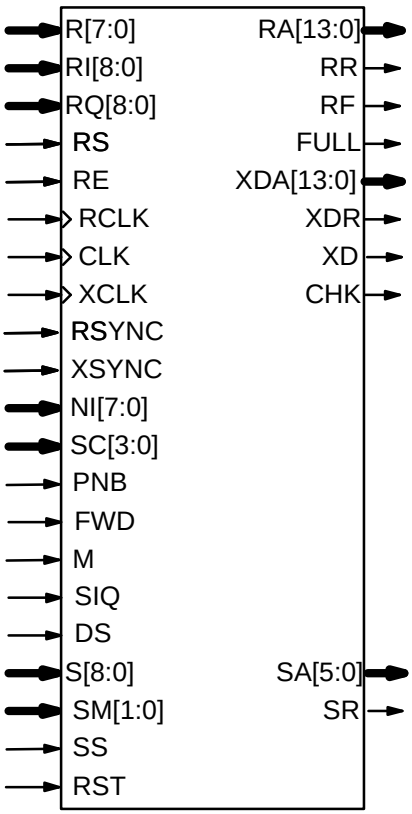

Figure 1: LCD01G schematic symbol.

The variable node degree is irregular with four or five different values for each code. Variable degrees range from 3 to 8 for the information bits. For the parity bits, due to the accumulate encoding method used, all column's have a variable degree of 2, except for the last column which has a degree of 1.

There are two sets of codes in the standard. PNB2(5,3) codes have a duration of 5 ms while PNB2(5,12) codes have a duration of 20 ms. For the PNB2(5,12) codes, no puncturing is used on the return link, while puncturing is used for the forward link. Three different modulation schemes are used; $\pi/4$ QPSK, 16APSK and 32APSK [2].

A modified Gauss–Seidel [3] iterative message passing algorithm is used in the decoder. In each clock cycle, M check nodes (where M is the circulant size) are decoded and the variable messages (VMs) updated. Each iteration requires qd_c clock cycles to calculate the updated VMs. We have that d_c is the larger check degree value and

q is the number of circulant rows. For calculation of the check messages (CMs), the scaled min–sum decoding algorithm [4] is used.

The LDPC decoder can achieve up to 115 Mbit/s with 30 iterations using a 200 MHz internal clock. Optional early stopping allows the decoder to reduce power consumption with no degradation in performance.

The decoder contains an input memory, VM memory, CM memory, VM sign memory, parameter memory and an output memory for the decoded data. The input and output memories are separated into two halves to allow ping–pong operation and maximum decoder speed. Separate clocks can be used for the input and output memories.

An optional demapper, descrambler and deinterleaver for QPSK, 16APSK or 32APSK modulation is included. This allows 9–bit inphase and quadrature samples to be demapped into individual bit log–likelihood ratios (LLRs), descrambled and deinterleaved.

Figure 1 shows the schematic symbol for the LCD01G decoder. The EDIF core can be used with Xilinx Vivado software to implement the core in Xilinx FPGA's. VHDL cores are used for Actel, Altera, Lattice and ASIC designs.

Table 1 shows the performance achieved with various Xilinx parts, 30 iterations and the slowest (PNB2(5,3) rate 4/5 C1) and fastest (PNB2(5,12) rate 9/10 C2) codes. T_{cp} is the minimum clock period over recommended operating conditions. These performance figures may change due to device utilisation and configuration.

Table 1: Performance of Xilinx parts.

Xilinx Part	T_{cp} (ns)	f_d (Mbit/s)	
		Low	High
XC5VLX85–1	6.596	19.1	87.4
XC5VLX85–2	5.635	22.4	102.4
XC5VLX85–3	5.000	25.3	115.4
XC6VLX75T–1	6.223	20.3	92.7
XC6VLX75T–2	5.290	23.9	109.0
XC6VLX75T–3	4.796	26.3	120.3
XC7A100T–1	6.652	19.0	86.7
XC7A100T–2	5.436	23.2	106.1
XC7A100T–3	4.800	26.3	120.2
XC7K70T–1	5.350	23.6	107.8
XC7K70T–2	4.349	29.1	132.6
XC7K70T–3	4.030	31.4	143.1

Signal Descriptions

CHK Parity Check
CLK System Clock
DS Descrambler Select
  0 = No descrambling
  1 = Descrambling
FWD Forward Link Select (PNB = 1 only)
  0 = No Depuncturing
  1 = Depuncturing
FULL Decoder Full (new data not accepted)
M Early Stopping Mode
  0 = No early stopping
  1 = Early stopping enabled
NI Number of Iterations minus one (0–63)
  I = NI+1 where *I* is number of iterations
PNB Packet Normal Burst Select
  0 = PNB2(5,3)
  1 = PNB2(5,12)
R Received LLR Data (8–bit)
RCLK Received Data Clock
RI Received Inphase Data (9–bit)
RI Received Quadrature Data (9–bit)
RA Received Data Address
RE Received Data Enable
RF Received Data Finish
RR Received Data Ready
RS Received Data Start
RST Synchronous Reset
RSYNC RCLK and CLK equal
S Symbol Value
SA Symbol Address
SC Code Select (0–8)
SIQ Inphase and Quadrature Select
  0 = No Demapping (data input to R)
  1 = Demapping (data input to RI and RQ)
SM0 Symbol Programming Modulation Select
  0 = 16APSK
  1 = 32APSK
SM1 Symbol Memory Select
  0 = ROM
  1 = RAM
SR Symbol Ready
SS Symbol Start
XCLK Decoded Data Clock
XD Decoded Data
XDA Decoded Data Address
XDR Decoded Data Ready
XSYNC XCLK and CLK equal

Table 2 lists the codes selected by SC for PNB2(5,3) codes (PNB = 0). Table 3 lists the codes selected by SC and FWD for PNB2(5,12)

codes (PNB = 1). The information data length is given by *K* and the coded data length after rate matching is given by *N*.

Table 2: PNB2(5,3) Codes (PNB = 0)

SC	Code	Modulation	*K*	*N*
0	1/2	QPSK*	488	958
1	2/3 C1	QPSK*	632	958
2	4/5 C1	QPSK*	760	958
3	9/10 C1	QPSK*	856	958
4	2/3 C2	16APSK	1272	1916
5	4/5 C2	16APSK	1528	1916
6	9/10 C2	16APSK	1720	1916
7	3/4	32APSK	1792	2395
8	4/5 C3	32APSK	1912	2395

* π/4–QPSK is transmitted, but must be derotated to QPSK for input to RI and RQ

Table 3: PNB2(5,12) Codes (PNB = 1)

SC	Code	Modulation	*K*	*N*	
				FWD=0	FWD=1
0	1/2	QPSK*	2208	4,440	4,344
1	2/3 C1	QPSK*	2960	4,440	4,344
2	4/5 C1	QPSK*	3552	4,440	4,344
3	9/10 C1	QPSK*	3992	4,440	4,344
4	2/3 C2	16APSK	5920	8,880	8,688
5	4/5 C2	16APSK	7104	8,880	8,688
6	9/10 C2	16APSK	7992	8,880	8,688
7	3/4	32APSK	8312	11,100	10,860
8	4/5 C3	32APSK	8880	11,100	10,860

* π/4–QPSK is transmitted, but must be derotated to QPSK for input to RI and RQ

LDPC Decoder Parameters

For binary modulation (which includes π/4 QPSK), we model the received sample at time *i* as

$$r_i = A(s_i + n_i) \quad (1)$$

where *A* is the no–noise amplitude, s_i is the modulated signal with value +1 for coded bit $y_i = 0$ and –1 for $y_i = 1$, n_i is additive white Gaussian noise (AWGN) with normalised variance

$$\sigma^2 = 1/(2RE_b/N_0) \quad (2)$$

and $R = K/N$ is the code rate.

The value of *A* directly corresponds to the 8–bit two's complement inputs for R (shown in Table 4). The 8–bit inputs have 256 quantisation regions. The quantisation regions are labelled from –128 to 127. For example, one could have *A* = 15.7. This value of *A* lies in quantisation region 16 (which has a range between 15.5 and 16.5).

Due to quantisation and limiting effects the value of *A* should be adjusted according to the received signal to noise ratio. We recommend *A* = 14, 17, 20 and 25 for rate 1/2, 2/3, 4/5 and 9/10 codes, respectively.

Table 4: Quantisation for received data R.

Decimal	Binary	Range
127	01111111	126.5↔∞
126	01111110	125.5↔126.5
⋮	⋮	⋮
2	00000010	1.5↔2.5
1	00000001	0.5↔1.5
0	00000000	–0.5↔0.5
–1	11111111	–1.5↔–0.5
–2	11111110	–2.5↔–1.5
⋮	⋮	⋮
–127	10000001	–127.5↔–126.5
–128	10000000	–∞↔–127.5

Demapping

For quadrature shift phase shift keying (QPSK) and amplitude phase shift keying (APSK) the two–dimensional (2–D) received signal is modelled by the inphase r_j^{I} and quadrature r_j^{Q} received values at symbol index *j*

$$r_j^{\mathrm{I}} = A(s_j^{\mathrm{I}} + n_j^{\mathrm{I}}) \quad (3)$$

$$r_j^{\mathrm{Q}} = A(s_j^{\mathrm{Q}} + n_j^{\mathrm{Q}}) \quad (4)$$

where s_j^{I} and s_j^{Q} are the inphase and quadrature components of the transmitted signal which has an average energy of 1 and n_j^{I} and n_j^{Q} are the inphase and quadrature AWGN, each with normalised variance

$$\sigma^2 = 1/(2sRE_b/N_0) \quad (5)$$

where *s* is the number of bits in each 2–D signal set (2 for QPSK, 4 for 16APSK and 5 for 32APSK).

For the inphase RI and quadrature RQ sampled values, 9–bit two's complement quantisation is used. Table 5 gives the quantisation for RI and RQ.

For best performance we recommend *A* = 191 for 16APSK and *A* = 194 for 32APSK. For QPSK modulation, we recommend *A* = 158, 192, 226 and 283 for rate 1/2, 2/3, 4/5 and 9/10 codes, respectively (these are the LLR values multiplied by $8\sqrt{2}$). Figures 2 and 3 show the signal sets within the 9–bit quantisation regions and recommended

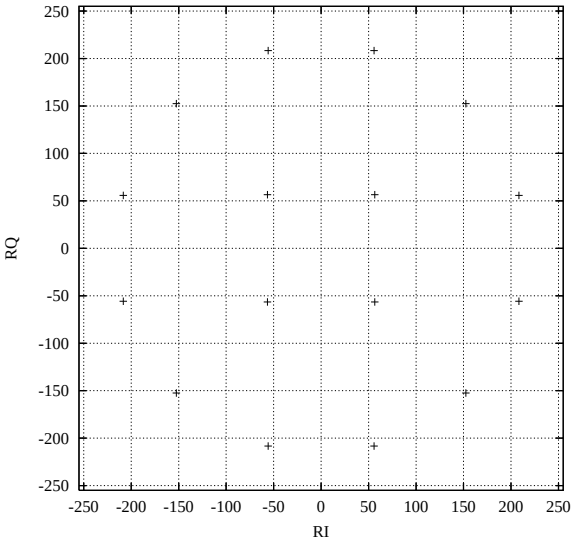

Figure 2: 16APSK Signal Set

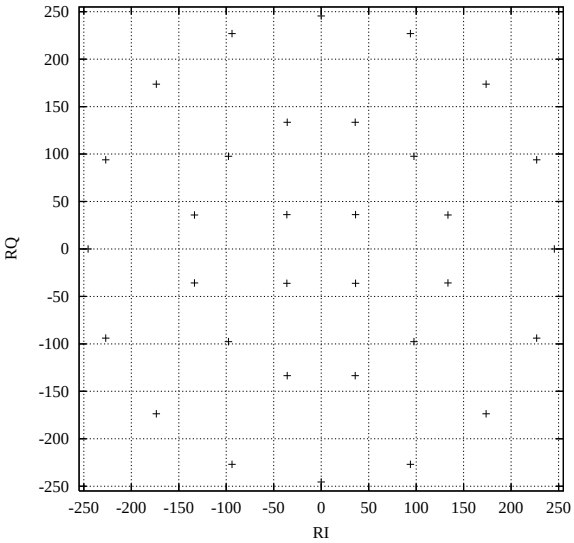

Figure 3: 32APSK Signal Set

signal amplitude for 16APSK and 32APSK, respectively.

Table 5: Quantisation for RI and RQ.

Decimal	Binary	Range
255	011111111	254.5↔∞
254	011111110	253.5↔254.5
⋮	⋮	⋮
2	000000010	1.5↔2.5
1	000000001	0.5↔1.5
0	000000000	–0.5↔0.5
–1	111111111	–1.5↔–0.5
–2	111111110	–2.5↔–1.5
⋮	⋮	⋮
–255	100000001	–255.5↔–254.5
–256	100000000	–∞↔–255.5

To demap the received APSK signal into the individual log–likelihood ratios (LLR) for each modulated bit, the optimum equation is

$$L_i = -\log\left(\frac{\sum_{h=0}^{S-1} h(i)\exp(-|s(h)-r|^2/c)}{\sum_{h=0}^{S-1} \overline{h(i)}\exp(-|s(h)-r|^2/c)}\right) \quad (6)$$

where i is the modulated bit index, $S = 2^s$ is the number of signal points, $h(i)$ is the i'th bit of h in binary, $s(h)$ is the h'th complex signal point, r is the received complex value and $c = 2A^2\sigma^2$. By using

$$\min^*(x,y) = \min(x,y) - c\ln(1+\exp(-|x-y|/c)) \quad (7)$$

where the Jacobian logarithm is to base $e^{1/c}$, the LLR can be simplified to

$$L_i = \min^*{}_{h=0}^{S-1} |s(h)-r|^2/h(i) - \min^*{}_{h=0}^{S-1} |s(h)-r|^2/\overline{h(i)}. \quad (8)$$

As the signal to noise ratio is high for APSK modulation, the correction term in (7) becomes negligible. The LLR can thus be simplified to

$$L_i \simeq \min_{h=0}^{S-1} |s(h)-r|^2/h(i) - \min_{h=0}^{S-1} |s(h)-r|^2/\overline{h(i)} \quad (9)$$

with little degradation in performance. The LLR is then scaled and rounded to the nearest integer. The result is an eight bit LLR which is deinterleaved for input to the decoder. Figure 4 gives an example plot of L_3 for 32APSK. In this case, the

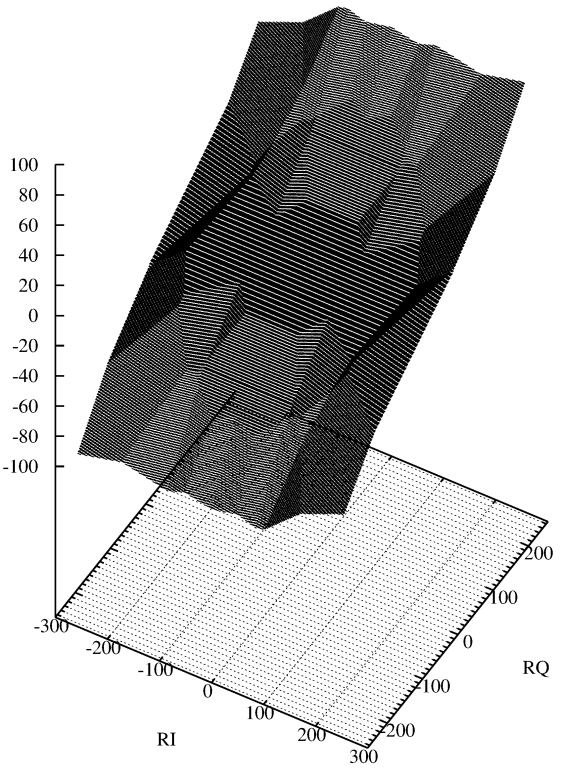

Figure 4: L_3 for 32APSK

LLR output was divided by 512 before rounding (giving a maximum amplitude of 85). For 16APSK we divide by 1024 (maximum amplitude of 46), except for rate 9/10, where we divide by 512.

Descrambling

After demapping, the 8–bit log–likelihood ratios must be descrambled [5]. The descrambler select DS input can be used to enable (DS=1) or disable (DS=0) descrambling.

The descrambler generator is a 15–bit right shift register with initial contents of 64531_8. The output is $e(D) = De(D) + D^{15}e(D) = e_0 + e_1D + e_2D^2 + ... + e_{N-1}D^{N-1}$. When $e_i = 1$ the LLR input has its sign inverted, otherwise the LLR is unchanged.

Deinterleaving

For 16APSK and 32APSK modulation, the descrambled LLRs must be deinterleaved. The interleaving operation consists of writing the N coded bits into length N/s columns and then reading the length s rows in an interleaved order which depends on the code used [1].

Deinterleaving is performed by writing the LLRs at the calculated interleaved address into the input RAM. For QPSK modulation, no interleaving is performed.

Demapping and deinterleaving is selected with SIQ = 1. Data is input to RI[8:0] and RQ[8:0]. This applies to both QPSK and APSK modulation. For SIQ = 0, the LLRs are directly input to R[7:0] of the decoder.

Decoder Speed

The number of LDPC decoder iterations is determined by NI, ranging from 0 to 255. NI = I–1 where I is the number of iterations. This is equivalent to 1 to 256 iterations. The decoder initially starts at iteration 0, increasing by one until NI is reached or an earlier time if early stopping is enabled.

The LDPC average decoder speed f_d is given by

$$f_d = \frac{F_d K}{Iqd_c + 4} \qquad (10)$$

where F_d is the CLK frequency, K is the input data length, I is the number of iterations (equal to NI+1), q is the number of circulant rows and d_c is the larger check degree value. Tables 6 and 7 give the performance of the decoder for PNB2(5,3) and PNB2(5,12) codes, respectively, F_d = 100 MHz and I = 30.

Table 6: PNB2(5,3) Decoder Speeds

SC	K	M	q	d_c	f_d (Mbit/s)
0	488	61	8	7	28.98
1	632	32	10	11	19.13
2	760	19	10	20	12.66
3	856	16	6	32	14.85
4	1272	53	12	12	29.42
5	1528	32	12	20	21.21
6	1720	32	6	37	25.81
7	1792	50	12	15	33.16
8	1912	48	10	20	31.85

Table 7: PNB2(5,12) Decoder Speeds

SC	K	M	q	d_c	f_d (Mbit/s)
0	2208	62	36	7	29.19
1	2960	74	20	11	44.82
2	3552	74	12	20	49.31
3	3992	64	7	37	51.35
4	5920	74	40	11	44.83
5	7104	74	24	20	49.32
6	7992	74	12	37	59.98
7	8312	87	32	15	57.71
8	8880	74	30	20	49.32

There are two decoder operation modes given by M. Mode M = 0 decodes a received block with a fixed number of iterations (given by NI). Mode M = 1 uses an early stopping algorithm. Early stopping is used to stop the decoder from iterating further once all the parity checks have been satisfied in the block. After the parity checks have been satisfied one extra iteration is performed.

Decoder Delay

The decoder delay can be separated into three parts. This is the input memory delay T_i, LDPC decoder delay T_l and the output memory delay T_o. Each delay is equal to

$$T_i = (14S_q + N + 2)T_r \qquad (11)$$

$$T_l = (Iqd_c + 5 - S_r)T_c \qquad (12)$$

$$T_o = (4 - 2S_x)T_x \qquad (13)$$

where T_r is the RCLK period, S_q = SIQ, S_r = RSYNC, T_c is the CLK period, T_x is the XCLK period and S_x = XSYNC.

The above delays assume that RE is high in the minimum time, no early stopping is used (M =

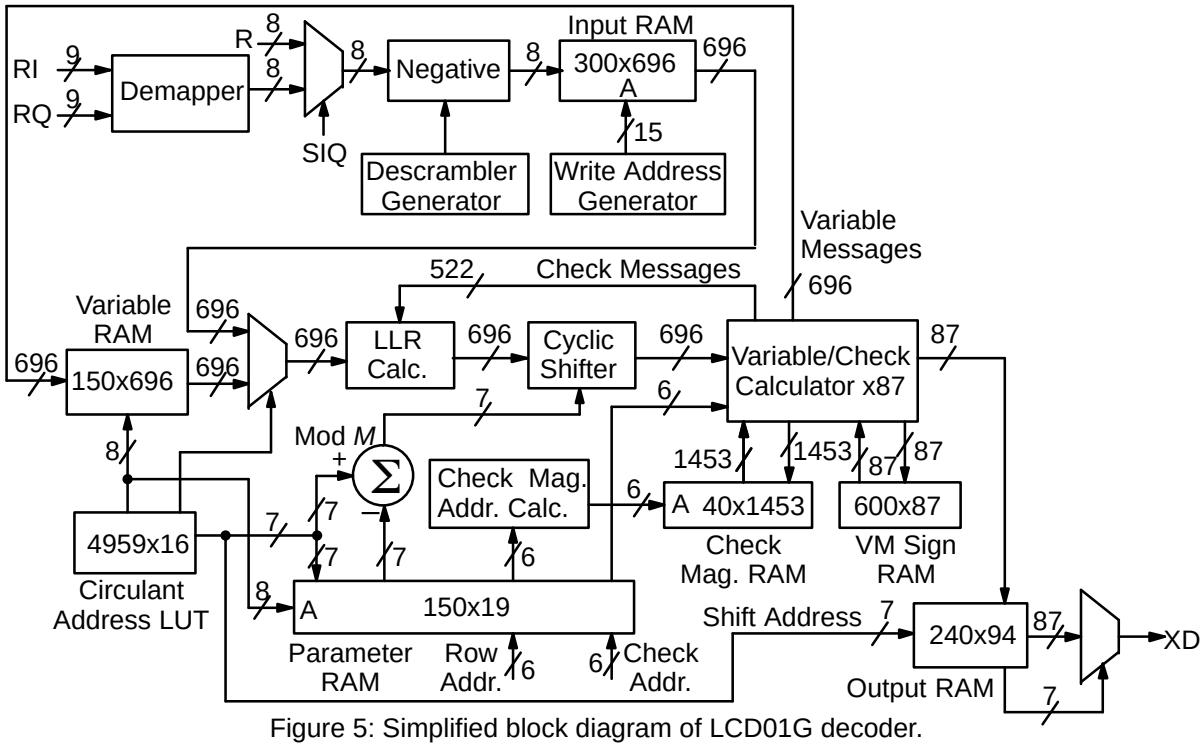

Figure 5: Simplified block diagram of LCD01G decoder.

0) and that the decoder parameters do not change between decoded blocks. When decoding multiple blocks with varying code parameters, the delay is more difficult to predict.

When RSYNC is low the actual LDPC decoder delay will vary from $T_I - T_C$ to T_I. Similarly, when XSYNC is low the actual output delay will vary from $T_O - T_X$ to T_O.

LDPC Decoder Operation

Figure 5 gives a simplified block diagram of the LCD01G decoder. Each memory is implemented using a number of smaller memories that cover the various code options.

The received or demapped input data are written 8–bits at a time into one half of the Input RAM. Input data corresponding to repeated bits are added together before writing. The other half of the memory has M 8–bit input data read into the decoder during the first iteration in each clock cycle. Input data corresponding to punctured bit positions are made equal to zero after reading.

The Circulant Address LUT is used to address the Input and Variable RAM, reading the circulants for each parity check equation. The Circulant Address LUT is also used to rotate the M LLRs using the cyclic shifter. As the new variable messages (VMs) are written in cyclic shifted order to reduce complexity, the current rotation is stored in the Parameter RAM. The next time that this circulant is read, the previously stored shift is subtracted from the current shift to derotate the previous shift. This technique can be used as the circulant row and column weight is one for all codes.

In each iteration, the VMs are read for d_c clock cycles and after a delay of d = 4 clock cycles, the new VMs are written for d_c clock cycles. At the same time as the new VMs are written, the VMs for the next circulants are read. The order in which the VMs are read and written is carefully arranged so that the VMs for each circulant are read after they are written.

As the largest value of M is 87, there are 87 parallel check/variable circuits. Each circuit serially inputs one of the M LLRs for d_c clock cycles. During this time, the previously stored VM sign bits are read for each of the LLR inputs.

The minimum magnitude, next minimum magnitude, VM parity and the address of the minimum magnitude value is read once at the beginning of the calculation. If the address matches the d_c clock address, then the next minimum magnitude is output, otherwise the minimum magnitude is output.

The VM parity, VM sign bit and selected magnitude are combined to form a two's complement CM which is subtracted from the LLR to form an 8–bit VM. This value is stored in the Variable RAM.

The d_c signs of the LLRs are serially exclusive–ORed (added modulo–2) to form the LLR parity check. The magnitude of the VMs (limited to 5–bits each) are used to serially find the minimum magnitude and next minimum magnitude using two comparator circuits. The d_c signs of the VMs

are serially exclusive–ORed together to calculate the VM parity.

The magnitudes are scaled by 0.78 or 0.91 for SIQ = 0 or 1, respectively, except for SC = 0 to 3 and 6 where 0.78 is always used. The magnitudes are then rounded down to the nearest integer. This avoids over optimistic values which reduces performance. Small lookup tables are used to perform the scaling. The two scaled magnitudes, VM parity and the address of the minimum magnitude are stored in the Check Magnitude RAM. This requires 17 bits each for the first 74 check circuits and 15 bits each for the remaining 13 check circuits. The VM sign bits are stored in the VM Sign RAM.

After the VMs have been read, the previously stored compressed CM values needs to be added to form the LLR. The VM parity is XORed with the sign bit of the VM to produce the sign bit of the CM. This is combined with the magnitudes to produce a two's complement CM. This CM is then added to the VM to produce the new LLR. To avoid over or under flow, positive CMs are added only if the VM is less than 192 and negative CMs are added only if the VM is greater than or equal to –192.

For the Check Magnitude RAM, there are three sets of read and write operations. There is one clock cycle to read the previous iteration CMs and one clock cycle to write the new CMs. This has to be done at the same time as reading d_c CMs for calculation of the new LLRs. However there are only d_c clock cycles to perform this. To reduce complexity, a single port RAM is used.

Let c be the number of circulants that have the same column address between rows. As $c \geq 1$ for accumulate type LDPC codes, this means that while writing the new CMs, the new CMs can be passed directly to the LLR calculator via a multiplexer. This operation is also performed for any common circulants between rows, since the new CMs have not yet been written into memory.

For the dual read operation, a single read of the same circulant is performed. We use the property that accumulate type LDPC codes have a variable degree of 2 for the check bits. That is, we read the circulant that is common with the next row at address d_c–1. This is used to calculate the CM to be added. The compressed CMs are then delayed and held for calculation of the CMs to be subtracted for the next row.

To ensure that the new VMs are written after the old VMs have been read, we must have $c \leq d_c–d$. This property is satisfied for all codes except the (958,488) code where c = 4 and $d_c–d$ = 3. To reduce c, the rows are permuted so that c is either 2 or 3.

Since the rows are permuted for the (958,488) code, the dual read property is lost. However, since c ≥ 2, this allow us to perform the CM read and write operation. For the other codes, c < 2 for some rows so this method can't always be used. Thus the the dual read method is used instead.

On the last iteration decoded data are stored in one half of the Output RAM at CLK, along with the current shift address. Address and multiplexer select generation are then used to read the Output RAM at XCLK so as to select the decoded output in the correct order.

The FULL output indicates when the decoder can accept data. When high this indicates that new data must not be input to the decoder in the next clock cycle. That is, RS and RE must remain low while FULL is high.

The received data start RS signal is used to start the decoder. The received data enable input RE must also go high when RS goes high to read the first received data. RE can only go high once for each received input symbol. For SIQ = 0 this means RE can only be high for N clock cycles. For SIQ = 1, RE goes high for a total of N/s clock cycles.

The received data ready output RR will go high to indicated that RE can now go high so as to read the next input. For SIQ = 0, RR will stay high until one clock cycle before the last data is input. For SIQ = 1, RR goes high s clock cycles after RE goes high, except after the last RE going high. RR will then stay high until after RE goes low. RS and RR can be ORed together to form RE if the input data is stored in an external memory.

Valid data must be input one clock cycle after RE goes high. A received data address output RA[13:0] is provided for reading received data from an external synchronous read input memory. Data read from the input memory must be held if RE goes low as shown in Figure 6 for SIQ = 0.

The received data finish output RF will go high for one clock cycle at the end of each received block. For SIQ = 0, this occurs when RA = N–1. For SIQ = 1, this occurs s–1 clock cycles after RA changes from N–1 to 0 at the end of the block.

If the other half of the Input RAM is available, FULL will remain low, indicating that the next block may be input. If both halves of the RAM are full, then FULL will go high. FULL will not go low again until one of the halves of the RAM becomes available. If FULL is low, RS and RE can go high.

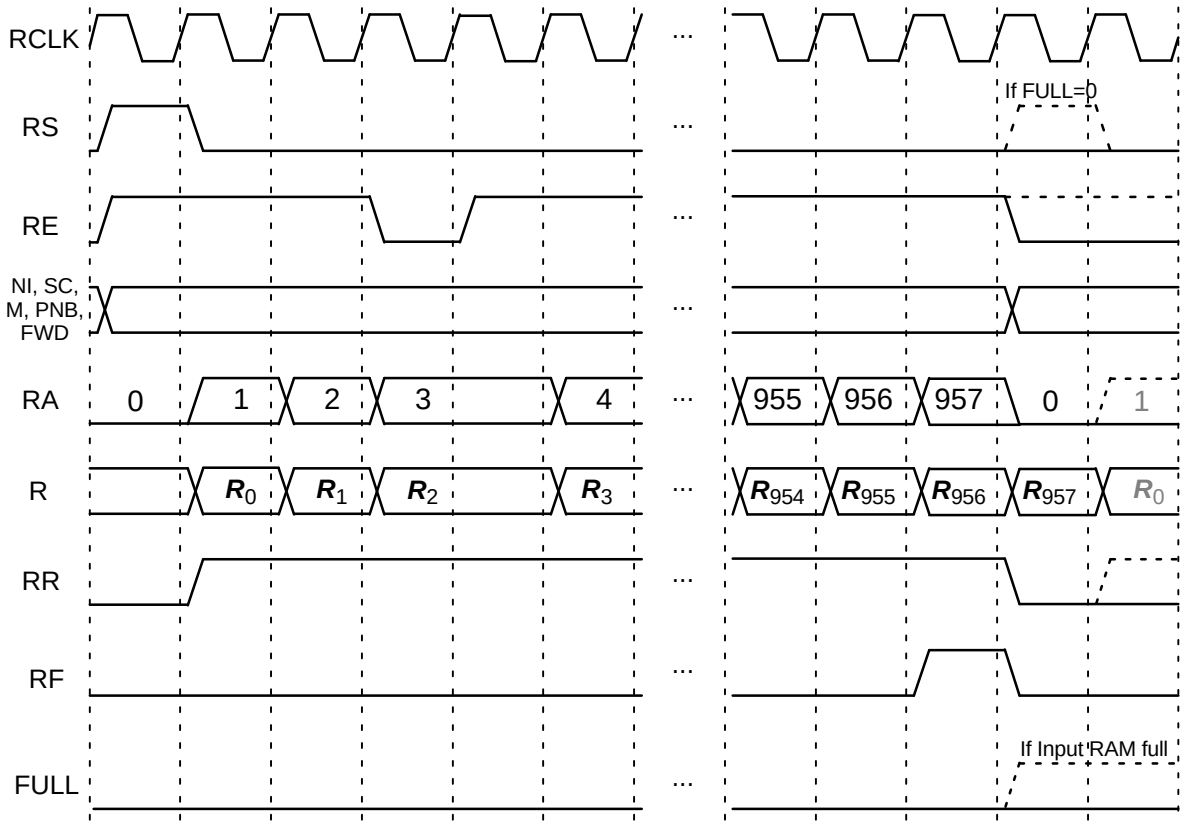

Figure 6: LDPC decoder input timing (SIQ = 0, PNB = 0, SC = 0).

Due to pipelining of the input data, FULL can can go high while data is being input. This can occur one or 15 clock cycles after RS goes high for SIQ = 0 and 1, respectively. If this occurs, RE must be held low and the input data R or RI and RQ held while FULL is high. When FULL goes low, data can then be continue to be input.

Inputs R[7:0], RI[8:0], RQ[8:0], RS, RE, NI[7:0], PNB, SC[3:0], FWD and M must be synchronous to RCLK. Outputs RA[13:0], RR, RF and FULL are synchronous to RCLK. Internal decoding uses CLK. If RCLK and CLK are equal to each other in both clock period and phase, then RSYNC can equal 1. This reduces the decoder input time by one clock cycle. If RCLK and CLK are not equal, then RSYNC must equal 0. RSYNC should not be connected to logic or input pins.

The input data can be input in any code order. That is, it is not necessary to wait for the decoder to output the last block of one code before changing to another code. If changing the code, the decoder parameters NI[7:0], SC[3:0], PNB, FWD and M must stay constant from the time RS goes high to until after RF goes high. Note that inputs SIQ and DS are not internally pipelined. These parameters can only be changed until after the last received block has been decoded.

Figure 6 illustrates the decoder input timing with SIQ = 0. Each received sample $\boldsymbol{R}_i$, $0 \le i \le N-1$ represents an 8–bit sample at time i. RS is shown going high again for the case where FULL = 0.

Figure 7 and 8 illustrates the decoder input timing with QPSK and 16APSK received data, respectively. Each received sample $\boldsymbol{R}_j$, $0 \le j \le N/s-1$ represents a two–dimensional sample at symbol time j. The RCLK frequency must be at least two, four or five times the symbol rate for QPSK, 16APSK and 32APSK, respectively. This is because the demapper calculates one LLR every RCLK cycle.

Figure 9 illustrates the decoder output timing. The decoded block is output from one half of the Output RAM after a block has been decoded. The signal XDR goes high for $K-1$ XCLK cycles while the block is output. The block is output in sequential order with address XDA[13:0].

Outputs XD, XDR, XDA[13:0] and CHK are synchronous to XCLK. If XCLK and CLK are equal to each other, i.e., they use the same clock signal, then XSYNC can equal 1. This reduces the decoder output time by two clock cycles. If XCLK and CLK are not equal, then XSYNC must equal 0. XSYNC should not be connected to logic or input pins.

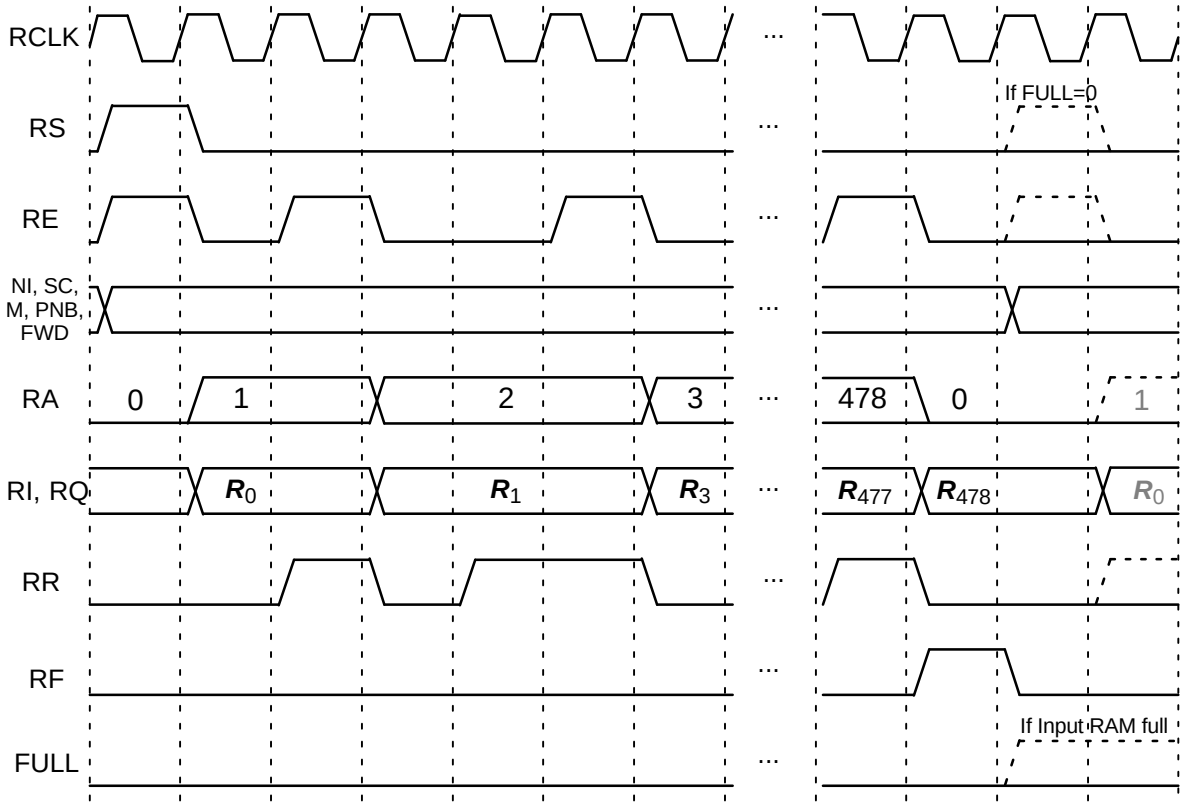

Figure 7: LDPC decoder input timing for QPSK (SIQ = 1, PNB = 0, SC = 0).

The early stopping algorithm uses the LLR parity checks to determine when to stop. If all the parity checks are satisfied, the decoder will continue for one more iteration and then stop decoding. Early stopping is selected with M = 1. If M = 0, all *I* iterations are performed. For high SNR

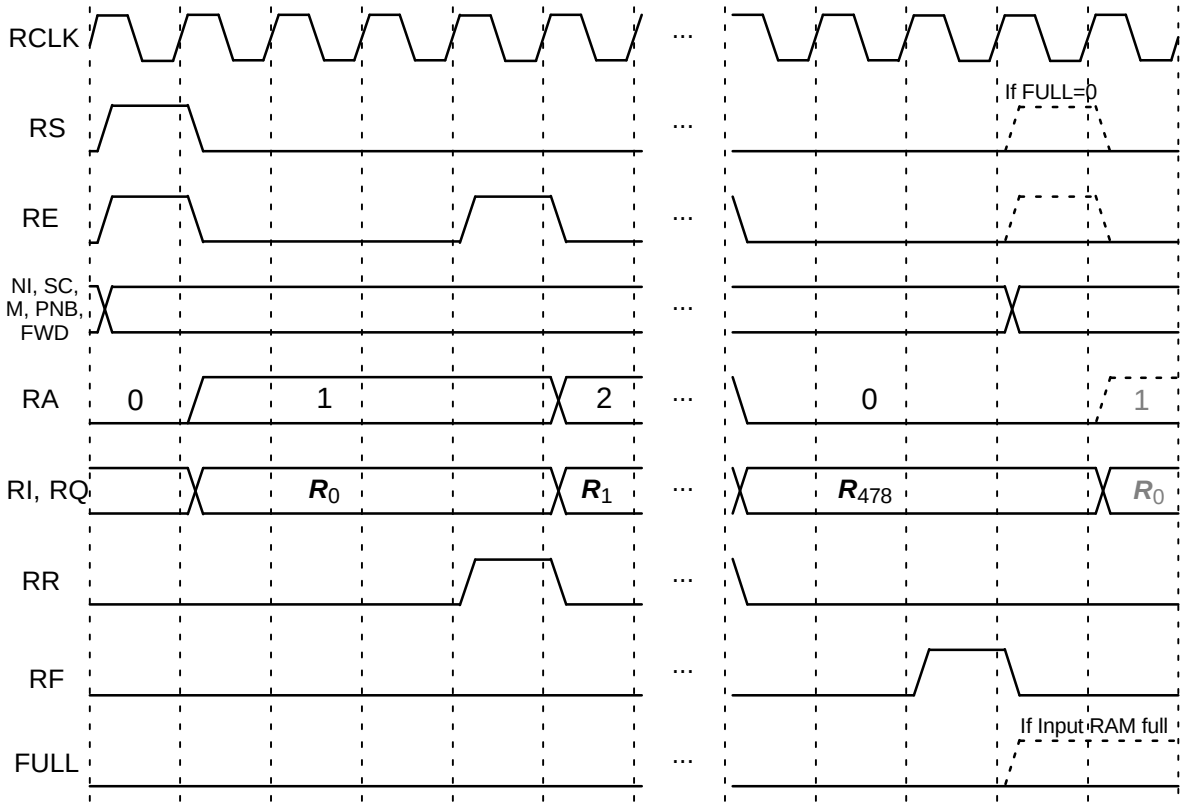

Figure 8: LDPC decoder input timing for 16QAM (SIQ = 1, PNB = 0, SC = 4).

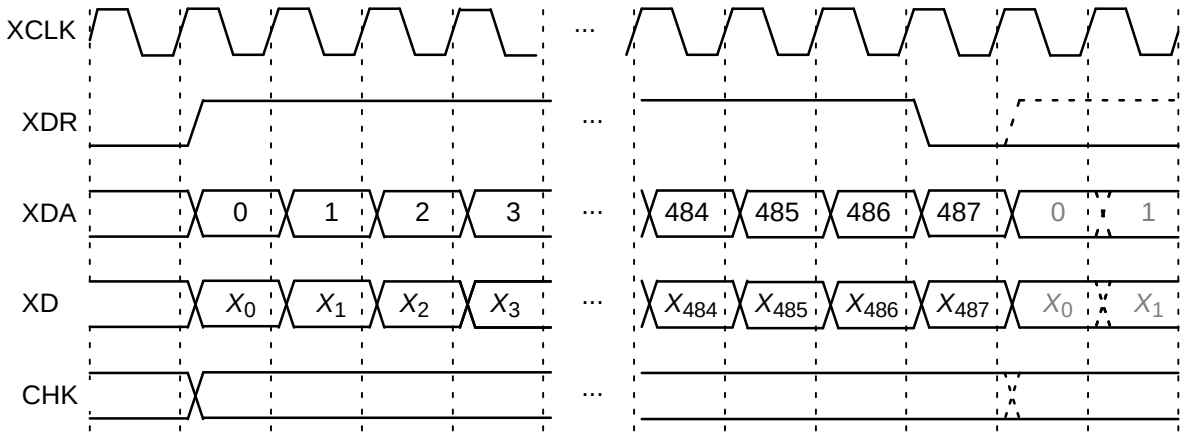

Figure 9: LDPC decoder output timing (PNB = 0, SC = 0).

operation early stopping can lead to significantly reduced power consumption, since most blocks will be decoded with a small number of iterations.

Parity Check Output

The CHK output provides an indication if the parity checks were satisfied during the last iteration. This output is valid while XDR is high. Note that the check is performed before the last LLR calculation. It is not performed on the decoded output. If CHK is high this indicates the parity checks were satisfied, indicating that there are probably no errors in the decoded data. If CHK is low, this indicates the checks were not satisfied and there are probably errors in the decoded data.

Computer simulations show that the probability of a missed detection, that is the proportion of frames that have errors where CHK = 1 (checks satisfied), is very low. We did not see any events of this type in our simulations. This means that if CHK = 1 it is very likely that there are no errors in the decoder data.

However, the probability of false detection, that is the proportion of frames that have no errors where CHK = 0 (checks not satisfied), can be high. For a low number of iterations or low SNR, the probability is very close to one. That is, nearly all frames that have no errors are falsely detected to have errors. As the number of iterations increases and the SNR increases the probability decreases to zero.

APSK Signal Set Programming

Each amplitude level in the APSK signal set can change amplitude and phase due to the effect of non–linear amplifiers. Thus, the LCD01G core is designed so that the signal set points can be programmed before the demapper can be used.

To start the programming sequence, SS should go high for one clock cycle. The address SA[5:0] will count from 0 to 2*S*–1, where *S* is the number of signal points (SM0 = 0 for *S* = 16 or SM0 = 1 for *S* = 32). SR will go high for 2*S*–1 clock cycles. The symbol data is input in alternating 9–bit two's complement values to S[8:0], with the inphase value first followed by the quadrature value. The external LUT or RAM should be read synchronously. SA1 corresponds to a_k in [2]. Figure 10 shows the programming sequence with SM0 = 0.

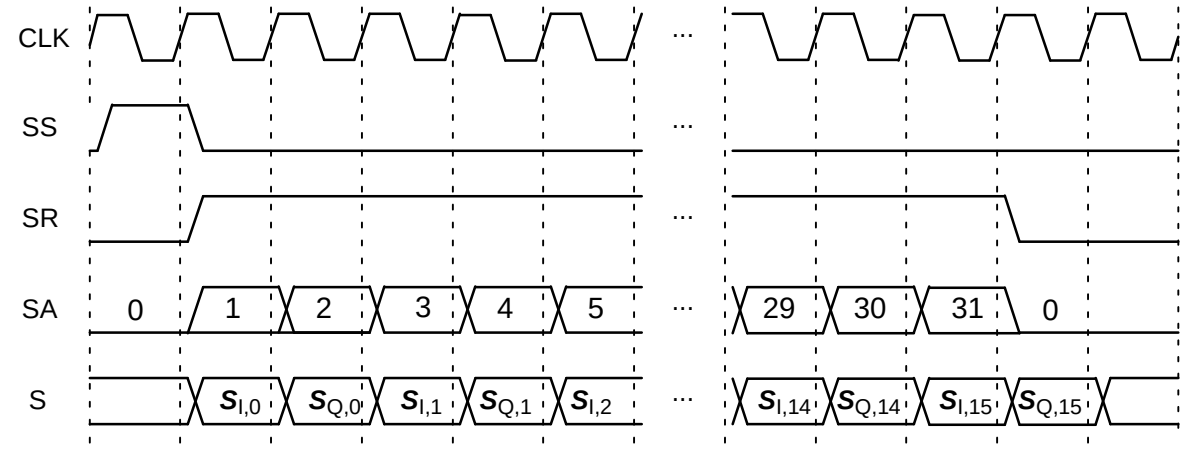

Figure 10: 16APSK signal set programming (SM0 = 0).

Input SM1 = 0 can be used to select an internal ROM with signal set positions corresponding to that in Figures 2 and 3. With SM1 = 1, the programmed RAM values are used. Note that SM1 is ignored during programming and SM0 is ignored during decoder operation.

Simulation Software

Free software for simulating the LCD01G LDPC decoder in additive white Gaussian noise (AWGN) or with external data is available by sending an email to info@sworld.com.au with "lcd01gsim request" in the subject header. The software uses an exact functional simulation of the LCD01G LDPC decoder, including all quantisation and limiting effects.

After unzipping lcd01gsim.zip, there should be lcd01gsim.exe, code.txt, ldpc.txt and apsk.txt. The files ldpc.txt and apsk.txt should not normally be edited, as they contain the LDPC code and APSK modulation parameters, respectively. The file code.txt contains the parameters for running lcd01gsim. These parameters are

`EbNomin`	Minimum E_b/N_0 (in dB)
`EbNomax`	Maximum E_b/N_0 (in dB)
`EbNoinc`	E_b/N_0 increment (in dB)
`A`	Binary modulation amplitude (1–31)
`ferrmax`	Number of frame errors to count
`Pfmin`	Minimum frame error rate (FER)
`Pbmin`	Minimum bit error rate (BER)
`state`	State file (0 to 2)
`s1`	Seed 1 (1 to 2147483562)
`s2`	Seed 2 (1 to 2147483398)
`q`	Number of quantisation bits (1 to 6)
`NI`	Number of iterations–1 (0 to 255)
`M`	Stopping mode (0 to 1)
`PNB`	Packet Normal Burst Select (0 or 1)
`SC`	Code Select (0–8)
`FWD`	Forward Link Select (PNB = 1 only, 0 or 1)
`DS`	Descrambler Select (0 or 1)
`SIQ`	Input Data Select (0 or 1)
`out_dir`	Output directory
`in_dir`	Input directory
`read_x`	Use external information data (y or n)
`read_r`	Use external received data (y or n)

The simulation will increase E_b/N_0 (in dB) in `EbNoinc` increments from `EbNomin` until `EbNomax` is reached or the frame error rate (FER) is below or equal to `Pfmin` or the bit error rate (BER) is below or equal to `Pbmin`. Each simulation point continues until the number of frame errors is equal to `ferrmax`. If `ferrmax` = 0, then only one frame is simulated.

When the simulation is finished the output is given in file k(*K*)(f).dat, for example k8880.dat where *K* = 8880 and FWD=0 or k8880f.dat where *K* = 8880 and FWD=1. The first line gives the E_b/N_0 (`Eb/No`), the number of frames (`num`), the number of bit errors in the frame (`err`), the total number of bit errors (`berr`), the total number of frame errors (`ferr`), the average number of iterations (`na`), the average BER (`Pb`) and the average FER (`Pf`). Following this, the number of iterations, `na`, `berr`, `ferr`, `Pb`, `Pf`, number of missed detections (`miss`), number of false detections (`fd`), missed detection rate (`Pmiss`) and false detection rate (`Pfd`) are given for each iteration.

The following file was used to give the BER for *K* = 488 in Figure 11. Figures 11, 12 and 13, show the BER for PNB2(5,3), PNB2(5,12) return link and PNB2(5,12) forward link, respectively. Auto–stopping was used. When iterating is stopped early, the `nasum` (`num*na`), `berr`, `ferr`, `miss` and `fd` results at stopping are copied for each iteration to the maximum iteration number. Thus, the *I* = 30 result is the performance one would measure with auto–stopping and *NI* = 29.

```
{EbNomin EbNomax EbNoinc A}
 1.0     5.0      0.1     158
{ferrmax Pfmin Pbmin}
 256     1e-99 1e-5
{state s1    s2}
 0     12345 67890
{q  NI  M  PNB  SC  FWD  DS  SIQ}
 8  29  1  0    0   0     1   1
{out_dir  in_dir  read_x  read_r}
 dat      input   n       n
```

The `state` input can be used to continue the simulation after the simulation has been stopped, e.g., by the program being closed or your computer crashing. For normal simulations, `state` = 0. While the program is running, the simulation state is alternatively written into state1.dat and state2.dat. Two state files are used in case the program stops while writing data into one file. To continue the simulation after the program is stopped follow these instructions:

1) Copy the state files state1.dat and state2.dat. This ensures you can restart the program if a mistake is made in configuring code.txt.

2) Examine the state files and choose one that isn't corrupted.

3) Change the state parameter to 1 if state1.dat is used or 2 if state2.dat is used.

4) Restart the simulation. The output will be appended to the existing k(*K*)(f).dat file.

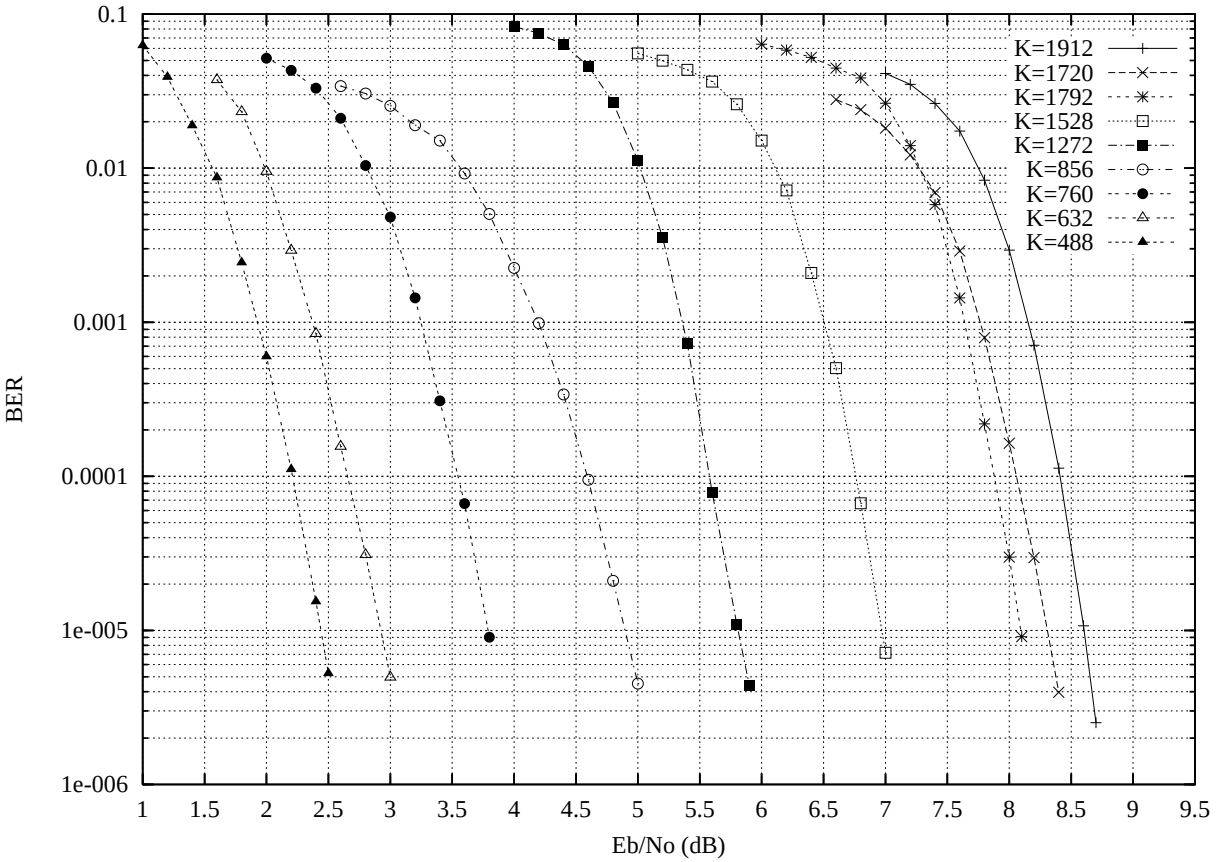

Figure 11: PNB2(5,3) BER performance with auto–stopping and 30 iterations.

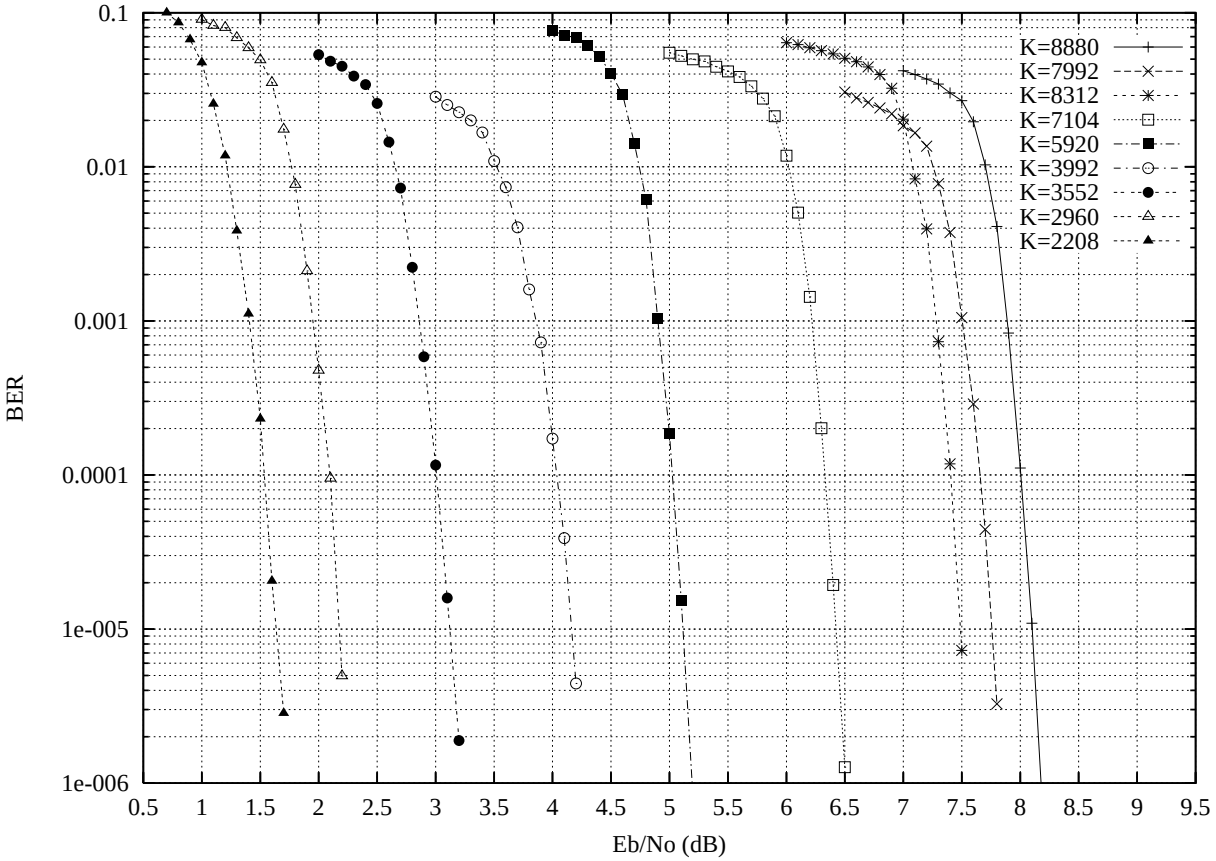

Figure 12: PNB2(5,12) return link BER performance with auto–stopping and 30 iterations.

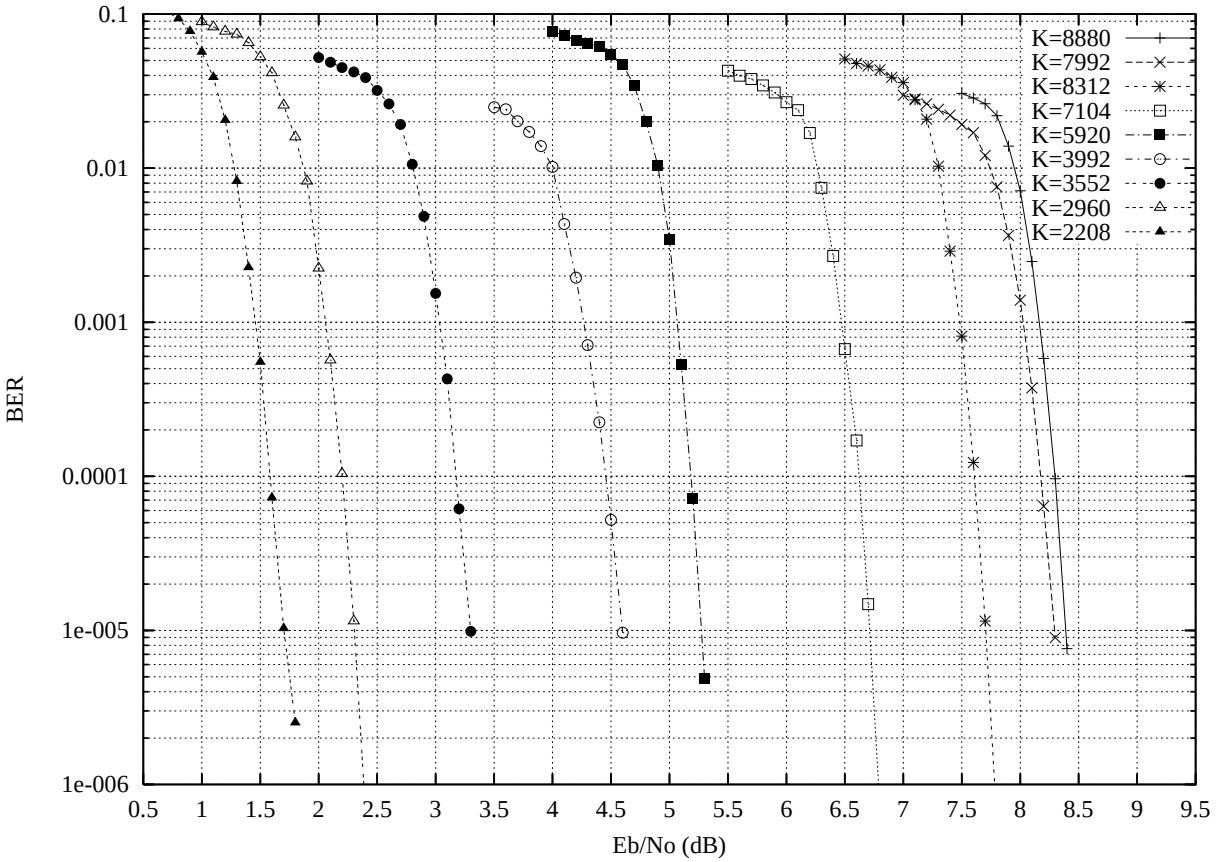

Figure 13: PNB2(5,12) forward link BER performance with auto–stopping and 30 iterations.

5) After the simulation has been completed, make sure that `state` is changed back to 0.

The software can also be used to encode and decode external data. To encode a block x_(*K*)(f).dat in the directory given by `in_dir`, set `read_x` to `y`, e.g., x_8880.dat in directory `input` (each line contains one bit of data). The encoded stream y_8880.dat will be output to the directory given by `out_dir`. If SIQ = 1, the modulated symbol output will be given in s_(*K*)(f).dat.

To decode data, place the received block of data in file r_(*K*)(f).dat in directory `in_dir` and set `read_r` to `y`. The decoded data is output to xd_(*K*)(f).dat in directory `out_dir`. For SIQ = 0, r_(*K*)(f).dat has in each line R[*i*], *i* = 0 to *N*–1 in decimal form, e.g., the first three lines could be

–25
9
31

For SIQ = 1, r_(*K*)(f).dat has in each line RI[*j*] and RQ[*j*], *j* = 0 to *N*/*s*–1 in decimal form, e.g., the first three lines could be

175 –140
155 97
89 106

LDPC Code Parameters

The LDPC code parameters are given in file ldpc.txt. Information in brackets {...} is a description of the data below. The first set of parameters are

k	Nominal data length
n	Nominal code length
c	Code value
kl	LDPC data length
nl	LDPC code length
xs	Number of shortened bits
xr	Number of repeated bits
xp	Number of punctured bits
M	Circulant size
q	Number of circulant rows
S	Number of points in signal set

The second set of parameters correspond to the variable degree values specified for the data

nv	Number of different variable degrees
dv(i)	Variable degree, i = 0 to nv–1
nv(i)	Number of degree dv(i) circulant col-

umns

The third set of parameters gives the addresses of the parity bit accumulators, obtained from [1].

The first nine codes correspond to PNB = 0 and SC = 0 to 8. The second nine codes correspond to PNB = 1. If desired, an additional nine codes can be appended to ldpc.txt by selecting PNB = 2, another nine codes with PNB = 3, etc.

APSK Modulation Parameters

The APSK modulation parameters are given in file apsk.txt. The first set of parameters are

S	Number of points in signal set
L	Number of different signal amplitudes
A	Average signal amplitude

The next three sets of parameters for level i = 1 to L are

n_i	Number of points for level i
r_i	Amplitude for level i
t_i	Initial phase in degrees for level i

The amplitudes are expressed as a ratio r_i/r_1. Thus only L–1 parameters are given for the amplitude.

The last set of parameters are the actual mappings. There are L rows, each row giving the decimal representation of the n_i points in each level. The first value in each row corresponds to the initial phase. The remaining points in each level are formed in anti–clockwise order.

The right most or least significant bit of the decimal value corresponds to bit a_k as described in [2]. If desired, the amplitudes and initial phase can be adjusted to reproduce the effects of a non–linear amplifier on the signal set. The value of A can also be adjusted to obtain the best performance. Other signal sets can be added, but there should be only one set of parameters for each S.

Ordering Information

SW–LCD01G–SOS (SignOnce Site License)
SW–LCD01G–SOP (SignOnce Project License)
SW–LCD01G–VHD (VHDL ASIC License)

All licenses include EDIF and VHDL cores. The VHDL cores can only be used for simulation in the SignOnce licenses. The SignOnce and ASIC licenses allows unlimited instantiations.

Note that *Small World Communications* only provides software and does not provide the actual devices themselves. Please contact *Small World Communications* for a quote.

References

[1] ETSI, “GEO–Mobile Radio interface specifications (Release 3); Third generation satellite packet radio service; Part 5: Radio interface physical layer specifications; Sub–part 3: Channel coding,” GMR–1 3G 45.003, ETSI TS 101 376–5–3 V3.3.1, Dec. 1012.

[2] ETSI, “GEO–Mobile Radio interface specifications (Release 3); Third generation satellite packet radio service; Part 5: Radio interface physical layer specifications; Sub–part 4: Modulation,” GMR–1 3G 45.004, ETSI TS 101 376–5–4 V3.3.1, Dec. 1012.

[3] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharam, “High throughput low–density parity–check decoder architectures,” *Global Telecommun. Conf.*, San Antonio, USA, vol. 5, pp. 3019–3024, Nov. 2001.

[4] J. Chen and M. P. C. Fossorier, “Near optimum universal belief propagation based decoding of low–density parity check codes,” *IEEE Trans. Commun.*, vol. 50, pp. 406–414, Mar. 2002.

[5] ETSI, “GEO–Mobile Radio interface specifications; Part 5: Radio interface physical layer specifications; Sub–part 3: Channel coding,” GMR–1 05.003, ETSI TS 101 376–5–3 V1.2.1, Apr. 2002.

Small World Communications does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its copyrights or any rights of others. *Small World Communications* reserves the right to make changes, at any time, in order to improve performance, function or design and to supply the best product possible. *Small World Communications* will not assume responsibility for the use of any circuitry described herein. *Small World Communications* does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. *Small World Communications* assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. *Small World Communications* will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

© 2013–2016 *Small World Communications*. All Rights Reserved. Xilinx and Virtex are regis-

tered trademark of Xilinx, Inc. All XC–prefix product designations are trademarks of Xilinx, Inc. All other trademarks and registered trademarks are the property of their respective owners.

Small World Communications, 6 First Avenue,
Payneham South SA 5070, Australia.
info@sworld.com.au ph. +61 8 8332 0319
http://www.sworld.com.au fax +61 8 8332 3177

Revision History

- v0.00 5 December 2013. Preliminary product specification.
- v0.01 6 June 2014. Increased maximum NI from 63 to 255. Added RSYNC and XSYNC inputs and BUSY output. Added Virtex–4 and Stratix–II complexity. Added Virtex–4 speed. Added start delay for decoder speed. Updated decoding speeds. Updated memory sizes and corrected shift address calculator in block diagram. Updated timing diagrams and associated text.
- v0.02 11 June 2014. Corrected code and decoder parameter change timing. Corrected file name description for lcd01gsim software.
- v0.03 28 August 2014. Changed R[5:0], APSK and SM inputs to R[7:0], SIQ and SM[1:0]. Changed BUSY output to FULL. Added DS input and RF output. Updated Xilinx and Altera resource utilisation, Xilinx Virtex–4 decoder speed, decoder block diagram, decoder speeds and input timing diagrams. Corrected decoder speed equation. Added descrambler, deinterleaver and decoder delay description. Added QPSK input timing diagram.
- v1.00 4 September 2014. First release. Added Virtex–5, Virtex–6, Artix–7, Kintex–7 and Zynq complexity and performance.
- v1.01 21 January 2015. Corrected amplitudes for SIQ = 1 and QPSK.
- v1.02 9 October 2015. Replaced deinterleaver RAM with write address generator. Corrected amplitudes for SIQ = 1 and QPSK.
- v1.03 21 January 2016. Changed decoding algorithm to modified Gauss–Seidel. Deleted Virtex 4 and Zync performance. Updated speed and BER performance.